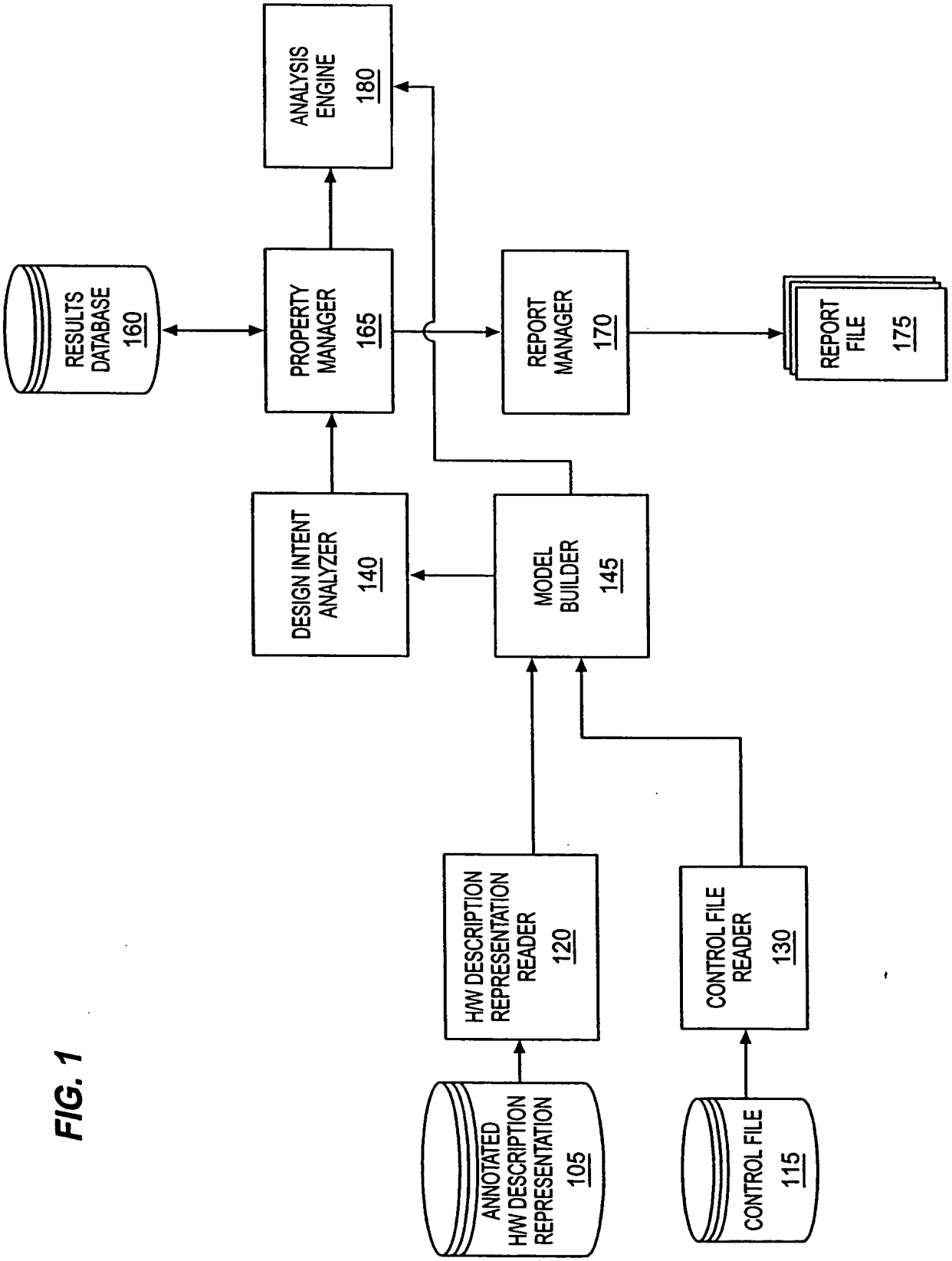
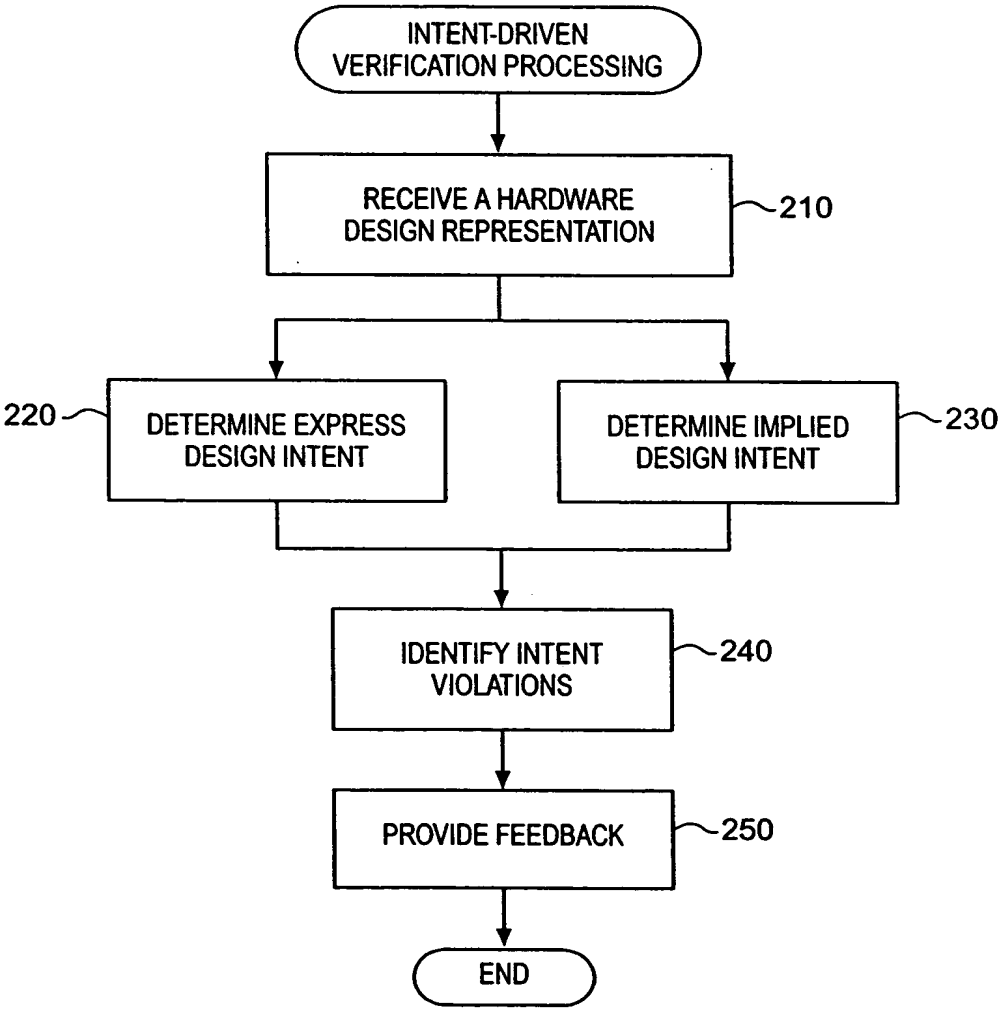

ANALYSIS ENGINE 180
RESULTS DATABASE 160
PROPERTY MANAGER 165
REPORT MANAGER 170
REPORT FILE 175
DESIGN INTENT ANALYZER 140
MODEL BUILDER 145
H/W DESCRIPTION REPRESENTATION READER 120
CONTROL FILE READER 130
ANNOTATED H/W DESCRIPTION REPRESENTATION 105
CONTROL FILE 115

FIG. 1

FIG. 2


INTENT-DRIVEN VERIFICATION PROCESSING
RECEIVE A HARDWARE DESIGN REPRESENTATION
210
220
DETERMINE EXPRESS DESIGN INTENT
DETERMINE IMPLIED DESIGN INTENT
230
IDENTIFY INTENT VIOLATIONS
240
PROVIDE FEEDBACK
250
END

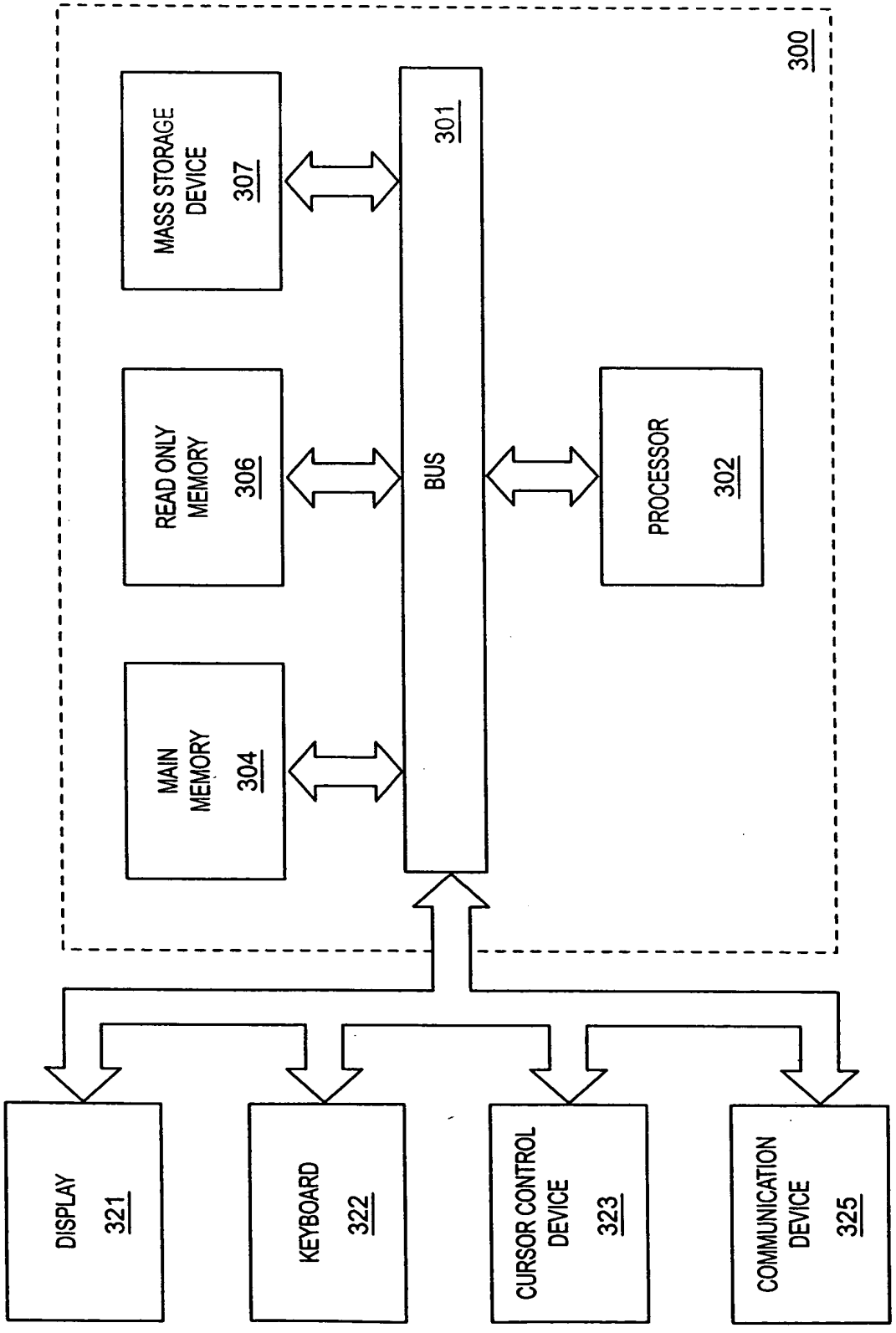

300
MASS STORAGE DEVICE 307
301
READ ONLY MEMORY 306
BUS
PROCESSOR 302
MAIN MEMORY 304
DISPLAY 321
KEYBOARD 322
CURSOR CONTROL DEVICE 323
COMMUNICATION DEVICE 325

FIG. 3

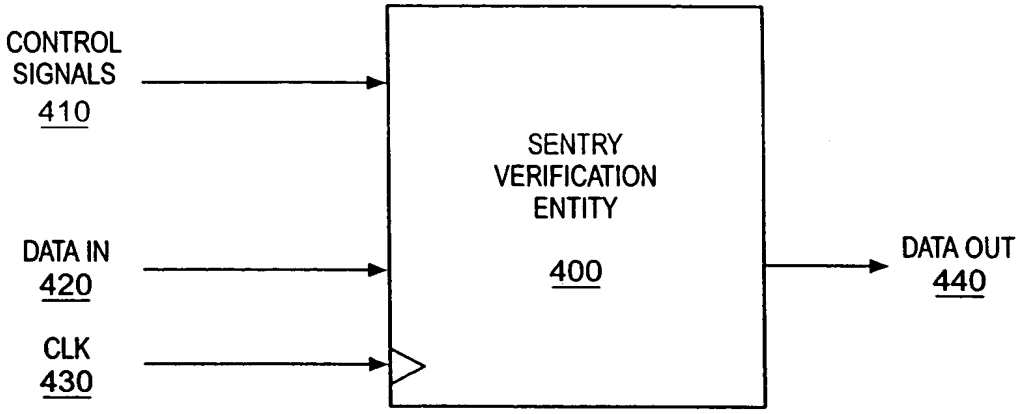

CONTROL SIGNALS
410
DATA IN
420
CLK
430
SENTRY VERIFICATION ENTITY
400
DATA OUT
440

FIG. 4A

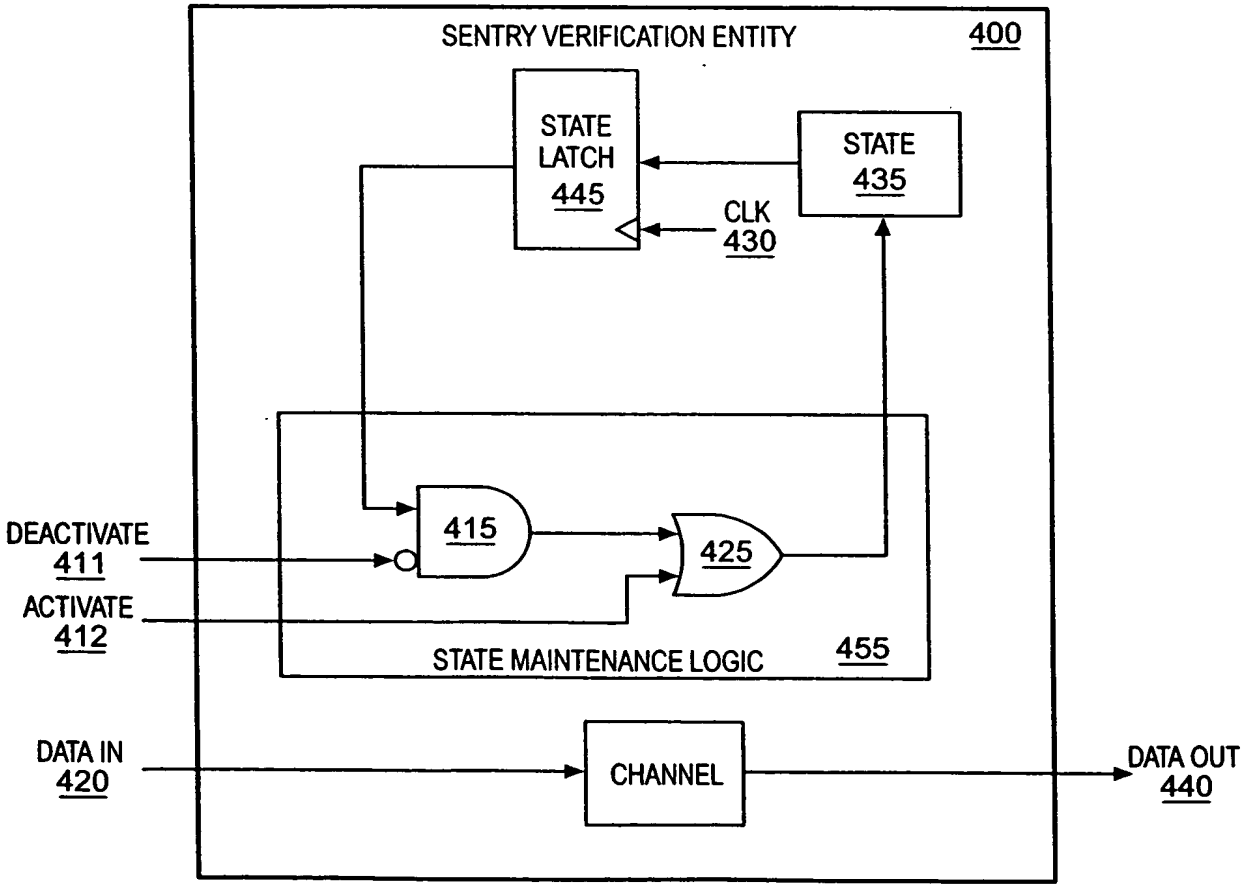

SENTRY VERIFICATION ENTITY
400
STATE LATCH
445
STATE
435
CLK
430
415
425
DEACTIVATE
411
ACTIVATE
412
STATE MAINTENANCE LOGIC
455
DATA IN
420
CHANNEL
DATA OUT
440

FIG. 4B

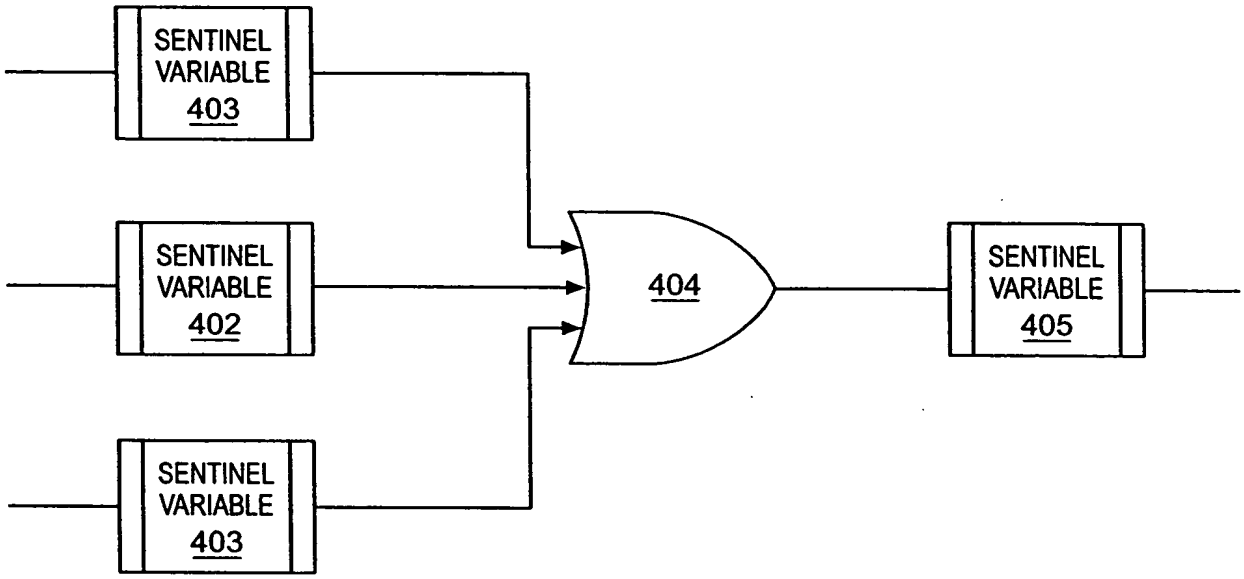
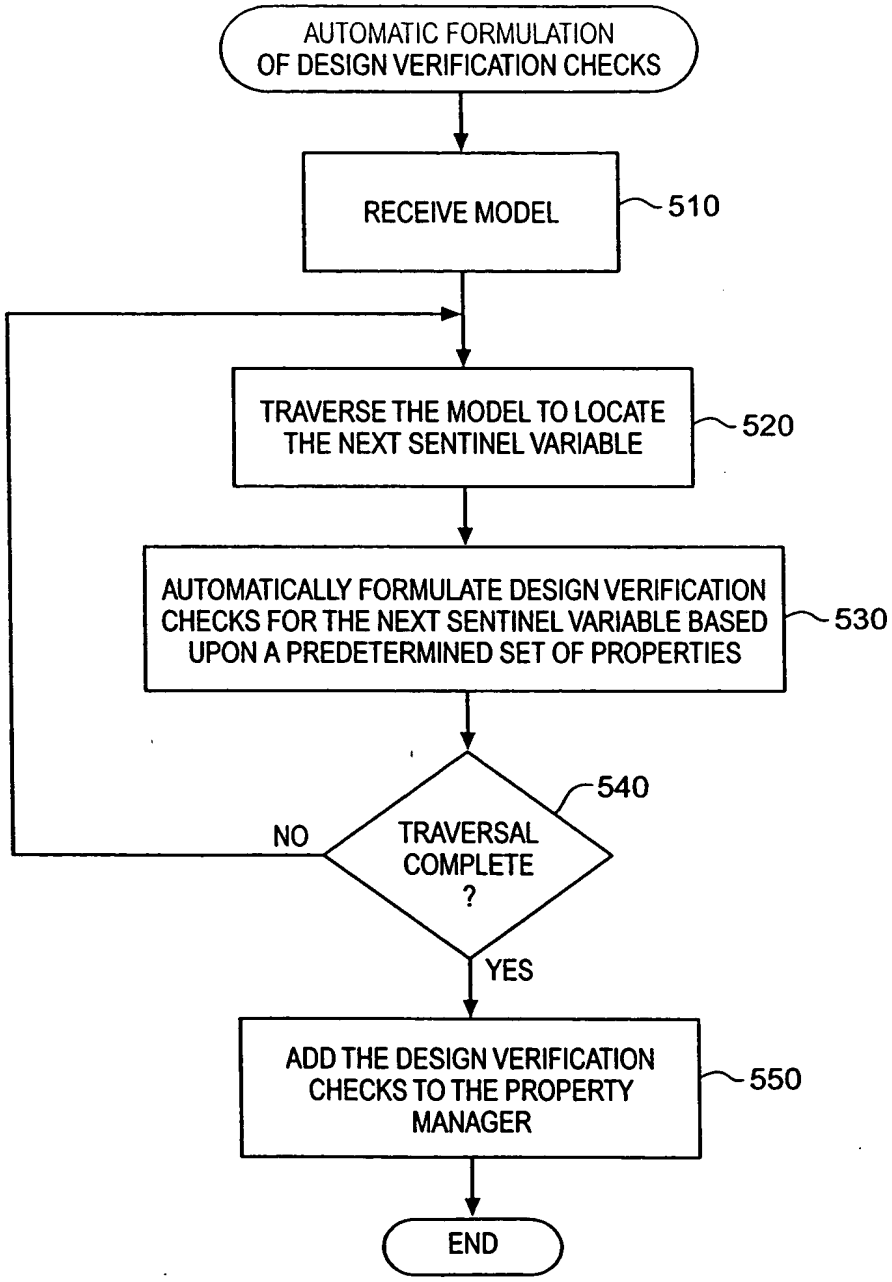

SENTINEL VARIABLE 403
SENTINEL VARIABLE 402
SENTINEL VARIABLE 403
404
SENTINEL VARIABLE 405

FIG. 4C

FIG. 5


AUTOMATIC FORMULATION OF DESIGN VERIFICATION CHECKS
RECEIVE MODEL
510
TRAVERSE THE MODEL TO LOCATE THE NEXT SENTINEL VARIABLE
520
AUTOMATICALLY FORMULATE DESIGN VERIFICATION CHECKS FOR THE NEXT SENTINEL VARIABLE BASED UPON A PREDETERMINED SET OF PROPERTIES
530
540
TRAVERSAL COMPLETE ?
NO
YES
ADD THE DESIGN VERIFICATION CHECKS TO THE PROPERTY MANAGER
550
END

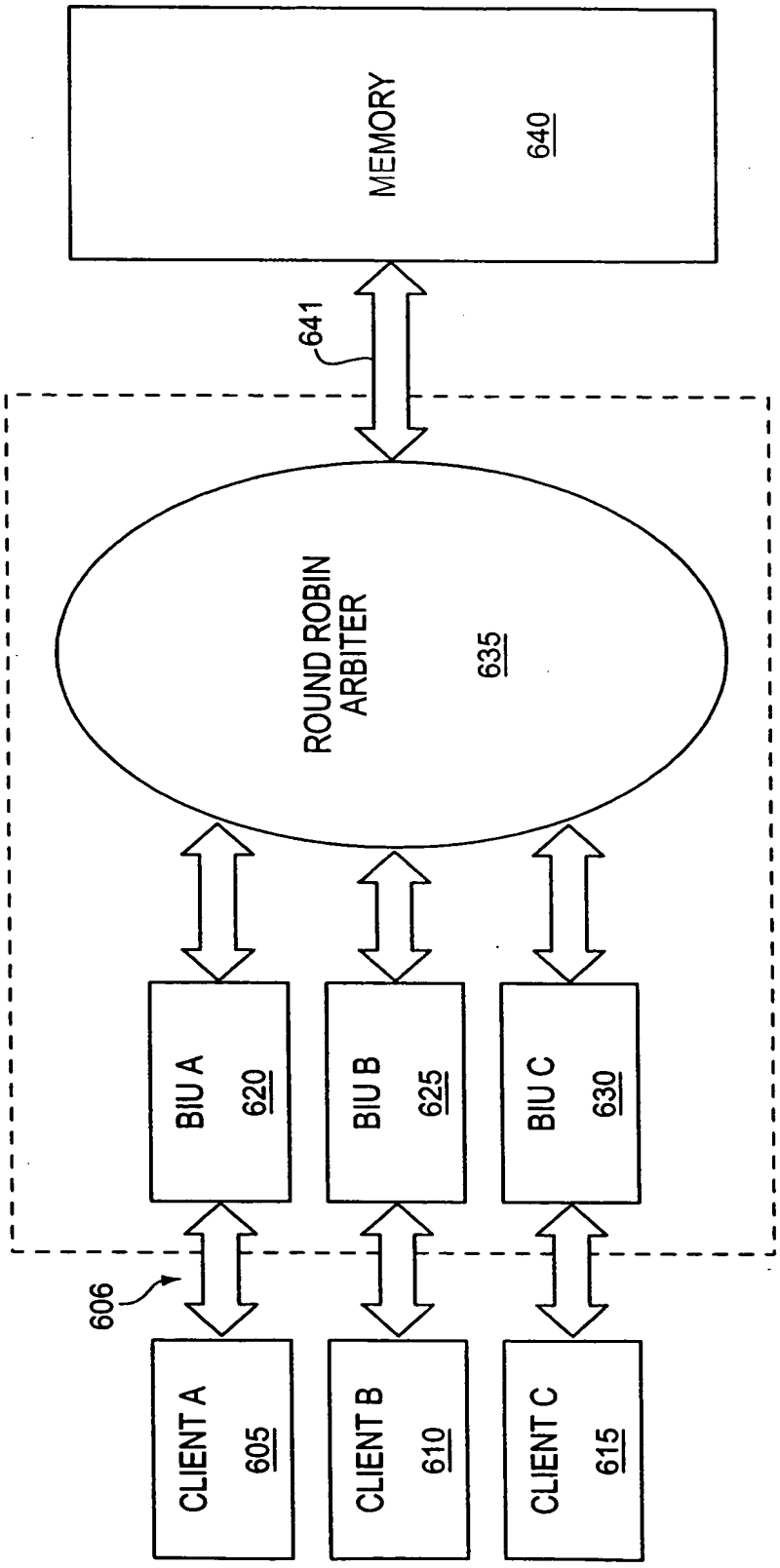

MEMORY
640
641
ROUND ROBIN ARBITER
635
BIU A
620
BIU B
625
BIU C
630
606
CLIENT A
605
CLIENT B
610
CLIENT C
615

FIG. 6A

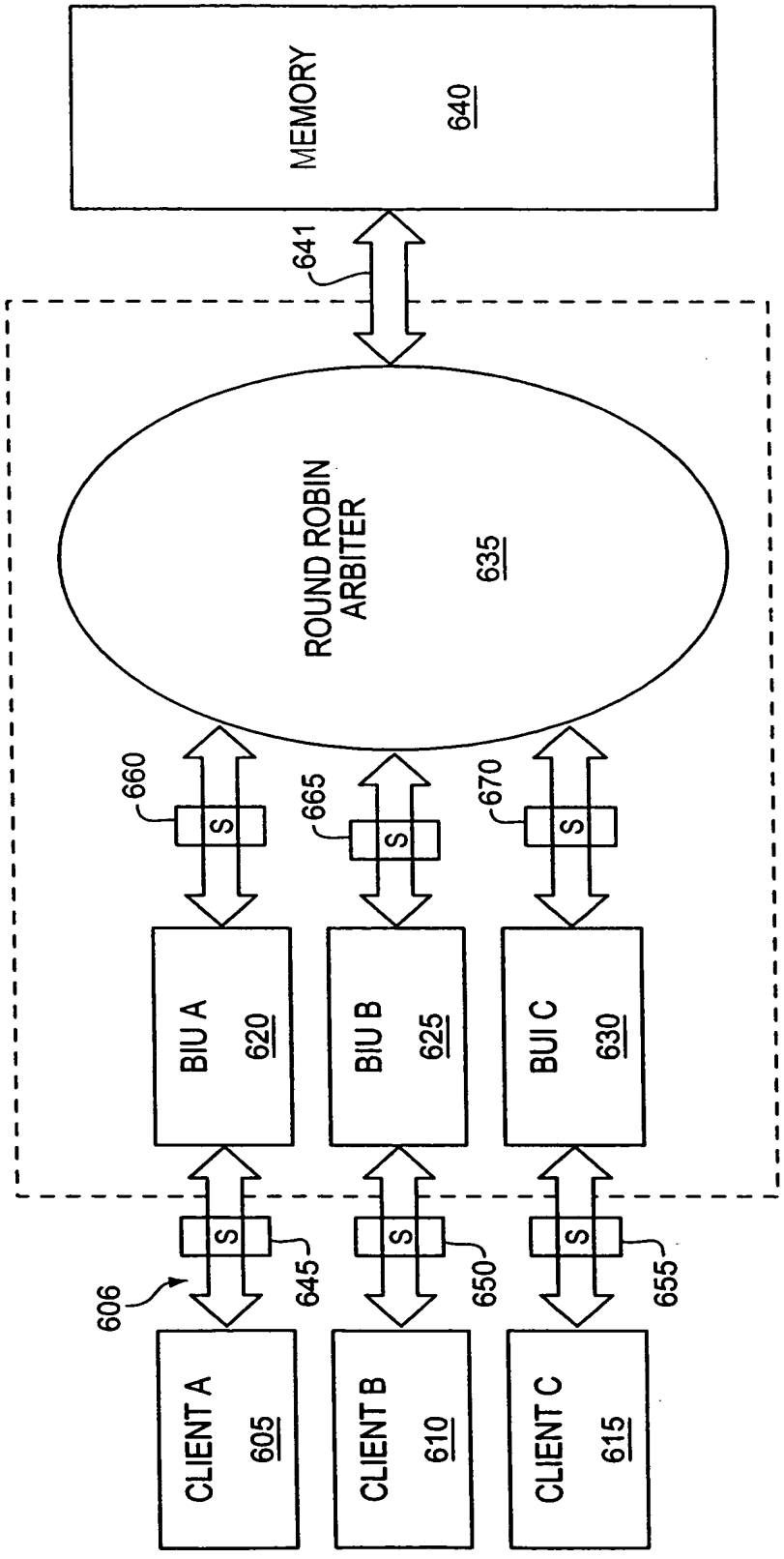

MEMORY
640
641
ROUND ROBIN ARBITER
635
660
665
670
S
BIU A
620
BIU B
625
BUI C
630
606
645
650
655
CLIENT A
605
CLIENT B
610
CLIENT C
615

FIG. 6B

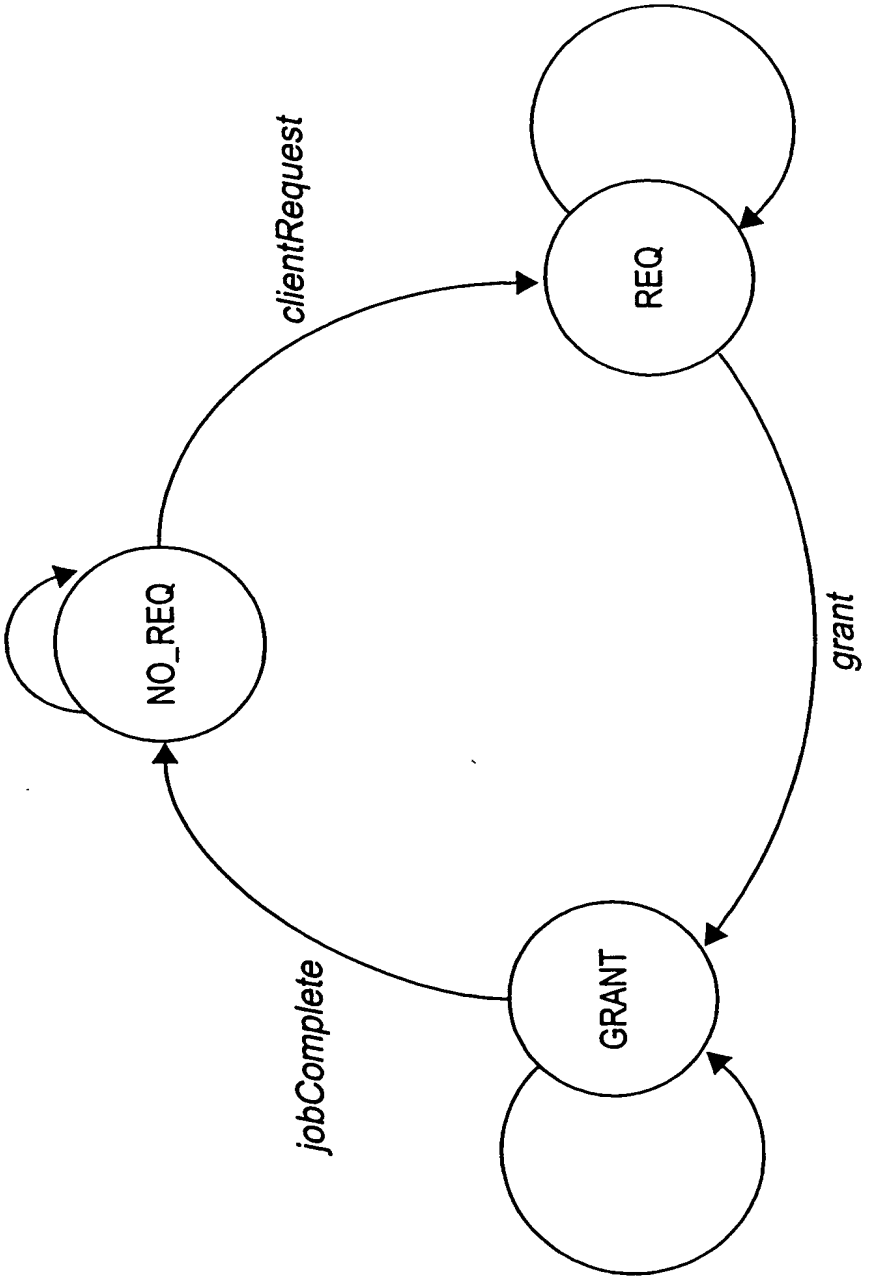


FIG. 7

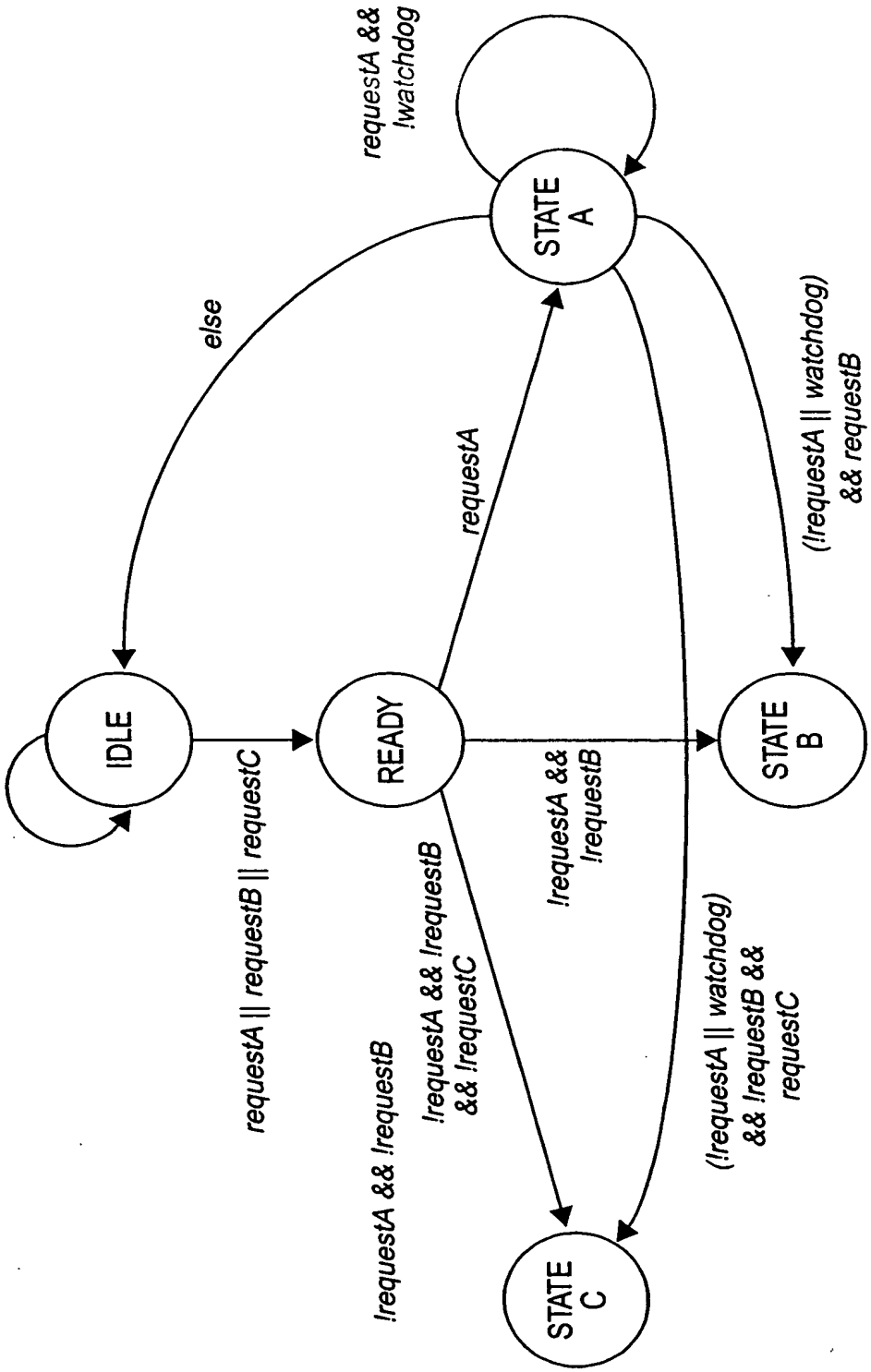

IDLE
READY
STATE A
STATE B
STATE C
requestA || requestB || requestC
!requestA && !requestB
requestA
else
requestA && !watchdog
!requestA && !requestB
!requestA && !requestB && requestC
(!requestA || watchdog) && !requestB && requestC
(!requestA || watchdog) && requestB

FIG. 8

```
1.  define DEFAULT_MAX_ACCESS_TIME 1'b1
2.
3.  module main(clk, reset,
4.          dataA, clientRequestA, jobCompleteA, clientGrantA,
5.          dataB, clientRequestB, jobCompleteB, clientGrantB,
6.          dataC, clientRequestC, jobCompleteC, clientGrantC,
7.          memoryData);
8.
9.          input       clk, reset;
10.         input [0:0] dataA, dataB, dataC;
11.         input       clientRequestA, clientRequestB, clientRequestC;
12.         input       jobCompleteA, jobCompleteB, jobCompleteC;
13.         output      clientGrantA, clientGrantB, clientGrantC;
14.         output [0:0] memoryData;
15.
16.         wire [0:0]  dataOutA, dataOutB, dataOutC;
17.
18.         // Put sentries at the boundary signals and make them active constantly.
19.
20.         // vx sentry dataA, dataB, dataC:clk;
21.         // vx always activate(dataA,dataB,dataC);
22.
23.         // Create three instances of the bus interface
24.
25.         BusInterface busInterfaceA(clk, reset, clientRequestA, JobCompleteA,
26.             dataA, dataOutA, requestA, grantA,
27.             clientGrantA);
28.
29.         BusInterface busInterfaceB(clk, reset, clientRequestB, jobCompleteB,
30.             dataB, dataOutB, requestB, grantB,
31.             clientGrantB);
32.
33.         BusInterface busInterfaceC(clk, reset, clientRequestC, JobCompleteC,
34.             dataC, dataOutC, requestC, grantC,
35.             clientGrantC);
36.
37.         RoundRobinArbiter arbiter(clk, reset,
38.             dataOutA, requestA, grantA,
39.             dataOutB, requestB, grantB,
40.             dataOutC, requestC, grantC,
41.             memoryData);
42.
43. endmodule // main
```

FIG. 9A

```
44.  module BusInterface(clk, reset, clientRequest, jobComplete,
45.          data, dataOut, request, grant, clientGrant);
46.
47.          input          clk, reset, clientRequest, jobComplete;
48.          input [0:0]    data;
49.          output [0:0]   dataOut;
50.          // vx sentry data0ut:clk;
51.
52.          output         request;
53.          input          grant;
54.          output         clientGrant;
55.
56.          parameter      NO_REQ = 2'bOO;
57.          parameter      REQ=2'b01;
58.          parameter      GRANTED = 2'b10;
59.
60.          reg [1:0] state, nextState;
61.          // vx flop state;
62.
63.          //assign request = ((state == REQ) || (state == GRANTED));
64.          assign request = ((state == REQ) || ((state == GRANTED) && !jobComplete));
65.          assign dataOut = data;
66.          assign clientGrant = grant;
67.
68.          always @(state or reset or clientRequest or jobComplete or grant)
69.          begin
70.                  if (reset)
71.                  begin
72.                          nextState = NO_REQ;
73.                          // vx deactivate(dataOut);
74.                  end
75.                  else
76.                  begin
77.                          nextState = state;
78.                          case (state)
79.                                  NO_REQ:
80.                                  begin
81.                                          // vx deactivate(dataOut);
82.                                          if (clientRequest) nextState = REQ;
83.                                  end
84.                                  REQ:
85.                                  begin
86.                                          // vx assert("env1", clientRequest && !jobComplete);
87.                                          if (grant) nextState = GRANTED;
88.                                  end
89.                                   VGRANTED:
90.                                  begin
91.                                          // vx activate(dataOut);
92.                                          // vx assert("env2", !clientRequest);
93.                                          if (jobComplete || 'grant)
94.                                                  nextState = NO_REQ;
95.                                  end
96.                          endcase // case(state)
97.                  end // else
98.          end // always
99.
100.         always @(posedge clk)
101.         state <= nextState;
102.  endmodule//BusInterface
```

FIG. 9B

```
103.  module RoundRobinArbiter(clk, reset,
104.          dataA, requestA, grantA,
105.          dataB, requestB, grantB,
106.          dataC, requestC, grantC,
107.          writeData);
108.
109.          input          clk, reset;
110.          input [0:0]    dataA, dataB, dataC;
111.          input          requestA, requestB, requestC;
112.          output         grantA, grantB, grantC;
113.          output [0:0]   writeData;
114.
115.          // vx sentry dataA, dataB, dataC: clk;
116.
117.          reg [0:0]      watchDogTimer, nextWatchDogTimer;
118.          reg [2:0]      state, nextState;
119.          // vx flop state, watchDogTimer;
120.          wire [0:0]     maxAccessTime;
121.
122.          parameter      idle = 3'bOOO;
123.          parameter      ready = 3'b001;
124.          parameter      stateA = 3'b010;
125.          parameter      stateB = 3'b011;
126.          parameter      stateC = 3'b100;
127.
128.          //next state logic
129.          always ©(reset or requestA or requestB or requestC or state or watchDogTimer)
130.          begin
131.                  // vx deactivate(dataA,dataB,dataC);
132.                  if (reset) begin
133.                          nextState= idle;
134.                          nextWatchDogTimer = 1 'b0;
135.          end
136.          else begin
137.                          next8tate= idle; // default state
138.                          //by default timer should increment
139.                          nextWatchDogTimer = watchDogTii-ner+1'bl;
```

FIG. 9C

```
140.
141.                    case (state)
142.                    idle:
143.                    begin
144.                            nextWatchDogTimer = 1'b0;
145.                            if (requestA || requestB || requestC) nextState = ready;
146.                    end
147.                    ready:
148.                    begin
149.                            if (requestA) nextState = stateA;
150.                            else if (requestB) nextState = stateB;
151.                            else if (requestC) nextState = stateC;
152.                            else nextState = idle;
153.                    end
154.                    stateA:
155.                    begin
156.                            // vx activate(dataA);
157.                            nextState= stateA;
158.                            // if request has been disabled or the max access time has
159.                            //reached change state.
160.                            if ((requestA == 1'b0) || (watchDogTimer == maxAccessTime)) begin
161.                                    nextWatchDogTimer = 1 'b0;
162.                                    if (requestB) nextState = stateB;
163.                                    else if (requestC) nextState = stateC;
164.                                    else nextState = idle;
165.                            end
166.                    end
167.                    stateB:
168.                    begin
169.                            // vx activate(dataB);
170.                            nextState= stateB;
171.                            if ((requestB == 1'b0) || (watchDogTimer == maxAccessTime)) begin
172.                                    nextWatchDogTimer = 1'b0;
173.                                    if (requestC) nextState= stateC;
174.                                    else if (requestA) nextState= stateA;
175.                                    else nextState= idle;
176.                            end
177.                    end
178.                    stateC:
179.                    begin
180.                            // vx activate(dataC);
181.                            nextState= stateC;
182.                            if ((requestC === 1'b0) || (watchDogTimer == maxAccessTime)) begin
183.                                    nextWatchDogTimer = 1'b0;
184.                                    if (requestA) nextState= stateA;
185.                                    else if (requestB) nextState= stateB;
186.                                    else nextState^ idle;
187.                            end
188.                    end
189.            endcase//case(state)
190.    end
191. end//always
```

FIG. 9D

```
192.        //state transition
193.        always @(posedge clk)
194.        begin
195.                state <= nextState;
196.                watchDogTimer <= nextWatchDogTimer;
197.        end
198.        //outputs
199.        assign grantA = (nextState == stateA);
200.        assign grantB = (nextState == stateB);
201.        assign grantC = (nextState == stateC);
202.        assign writeData = (grantA ? dataA : 'DATA_WIDTH'bz);
203.        assign writeData = (grantB ? dataB : 'DATA_WIDTH'bz);
204.        assign writeData = (grantC ? dataC : 'DATA_WIDTH'bz);
205.        assign maxAccessTime = •DEFAULT_MAX_ACCESS_TIME;
206.        // vx always onetrue("grant", grantA, grantB, grantC);
207.  endmodule // RoundRobinArbiter
```

FIG. 9E

FIG. 10

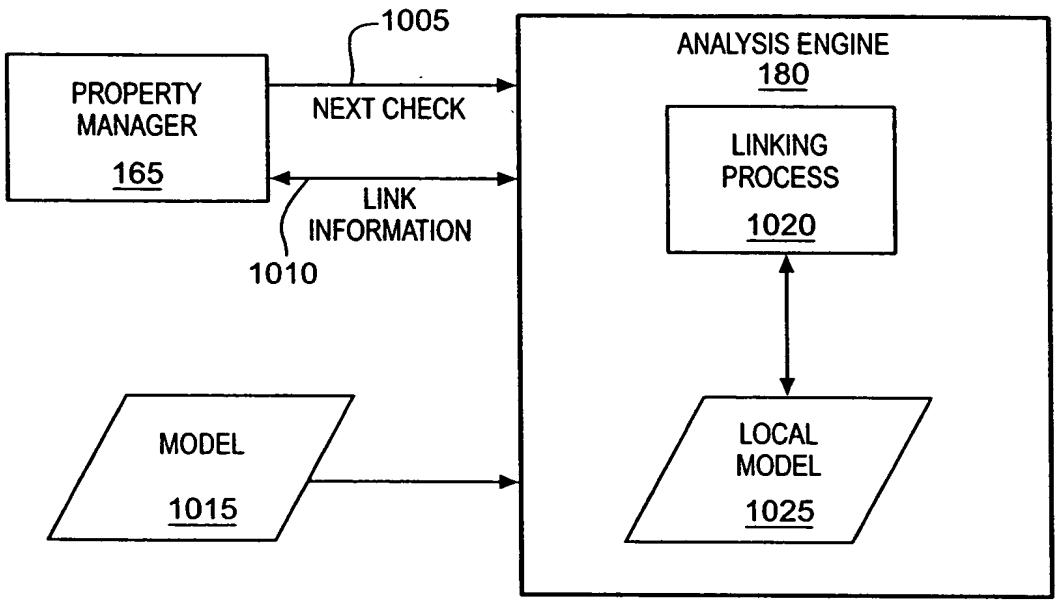

1005
PROPERTY MANAGER
165
NEXT CHECK
LINK INFORMATION
1010
ANALYSIS ENGINE
180
LINKING PROCESS
1020
MODEL
1015
LOCAL MODEL
1025

FIG. 11

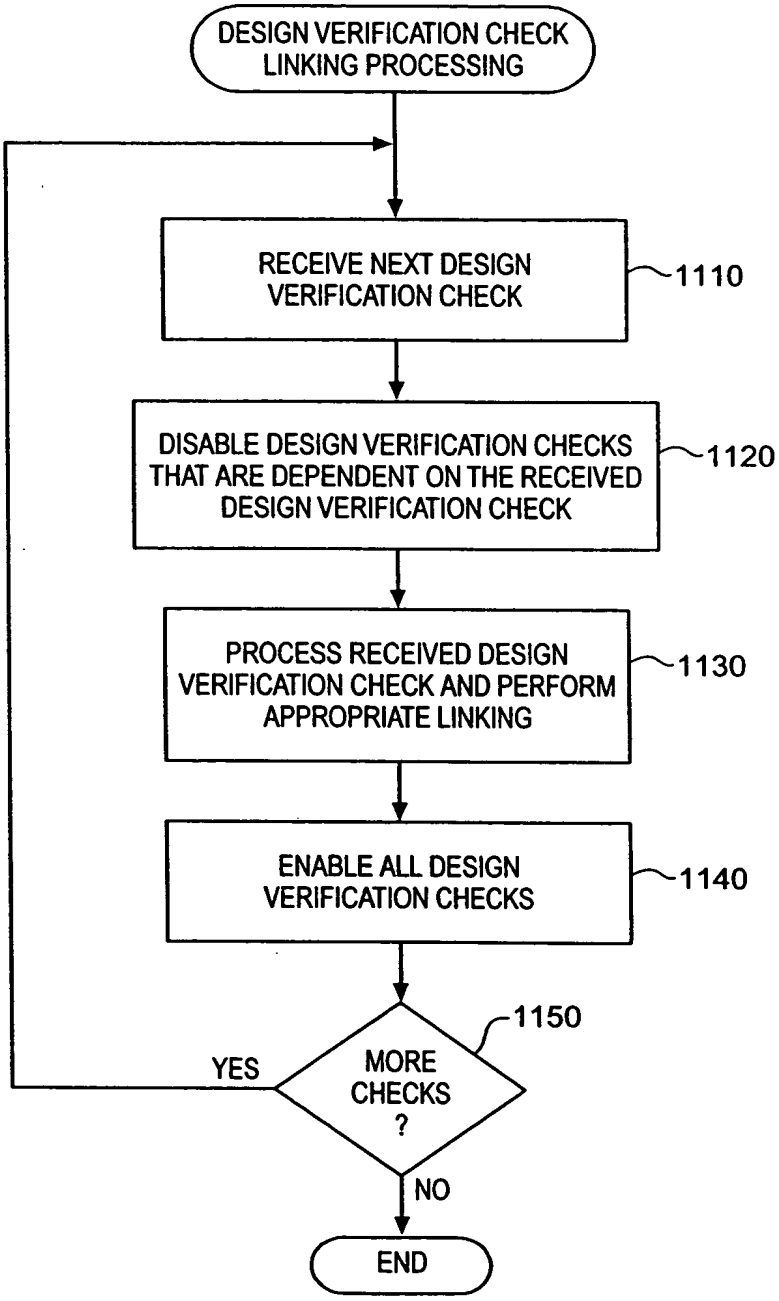
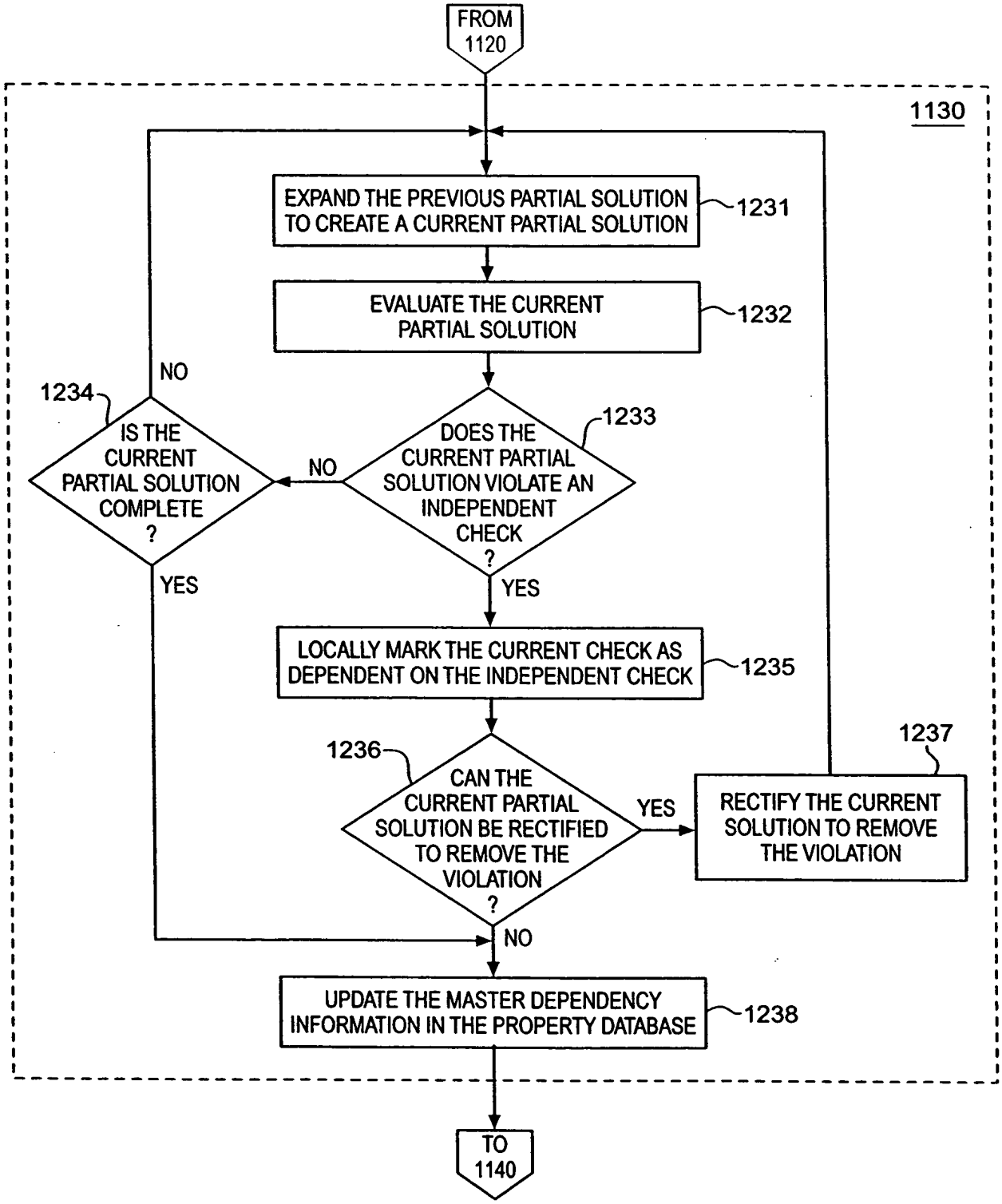

DESIGN VERIFICATION CHECK LINKING PROCESSING
RECEIVE NEXT DESIGN VERIFICATION CHECK
1110
DISABLE DESIGN VERIFICATION CHECKS THAT ARE DEPENDENT ON THE RECEIVED DESIGN VERIFICATION CHECK
1120
PROCESS RECEIVED DESIGN VERIFICATION CHECK AND PERFORM APPROPRIATE LINKING
1130
ENABLE ALL DESIGN VERIFICATION CHECKS
1140
MORE CHECKS ?
1150
YES
NO
END

FIG. 12


FROM 1120
1130
EXPAND THE PREVIOUS PARTIAL SOLUTION TO CREATE A CURRENT PARTIAL SOLUTION
1231
EVALUATE THE CURRENT PARTIAL SOLUTION
1232
1234
NO
IS THE CURRENT PARTIAL SOLUTION COMPLETE ?
1233
DOES THE CURRENT PARTIAL SOLUTION VIOLATE AN INDEPENDENT CHECK ?
NO
YES
YES
LOCALLY MARK THE CURRENT CHECK AS DEPENDENT ON THE INDEPENDENT CHECK
1235
1236
CAN THE CURRENT PARTIAL SOLUTION BE RECTIFIED TO REMOVE THE VIOLATION ?
YES
1237
RECTIFY THE CURRENT SOLUTION TO REMOVE THE VIOLATION
NO
UPDATE THE MASTER DEPENDENCY INFORMATION IN THE PROPERTY DATABASE
1238
TO 1140

```
1     ====================================
2.    Validation Results for Module "main"
3     ====================================
4.
5     ========================
6.    Functional Checks Summary
7.    ========================
8.    Failed checks (FAILED)                                    = 5
9.    Bounded failed checks (BOUNDED_FAIL)                      = 0
10.   Inconclusive checks (INCONCLUSIVE)                        = 3
11.   Secondary failed checks (SECONDARY)                       = 0
12.   Interface checks (INTERFACE)                              = 12
13.   Skipped checks (UNPROCESSED/DISABLED/DEFERRED)            = 0
14.   Bounded passed checks (BOUNDED_PASS)                      = 0
15.   Passed checks (PASSED/USER PASSED/CONDITIONAL)            = 144
16.   ------------------------------------
17.   Total checks                                              =164
18.
19    ==========================
20.   RTL Characteristics Summary
21.   ==========================
22.   Number of inferred flops              = 10
23.   Number of inferred latches            = 0
24.   Number of static X sources            = 0
25.   Number of non-resettable flops        = 0
26.
27    ================================
28.   Summary of failed checks by type:
29.   ================================
30.   [CA] =0   (out of 39)
31.   [BE] = 1   (out of 40)
32.   [AX] = 1   (out of 15)
33.   [CME]=0   (out of 10)
34.   [CV] =0   (out of 29)
35.   [AC] =0   (out of 7)
36.   [AAO]=0   (out of 6)
37.   [AID]=0   (out of 9)
38.   [LVD] = 3   (out of 9)
```

FIG. 13A

```
39.
40.   Block enable [BE] : 1 failed check
41.   ----------------------------------
42.   FAILED    BE   : arbiter.nextState_bit0_AX_6
43.                  : Enable condition for block with assignment "arbiter.nextState_bit0_AX_6" is always off
44.                  : Block with assignment "arbiter.nextState_bit0_AX_6" is defined on line 191 of "rr3.v"
45.
46.   Assignment execution [AX]  : 1 failed check
47.   ------------------------------------------
48.   FAILED    AX   : arbiter.nextWatehDogTimer_bit0_AX_1
49.                  : Assignment "arbiter.nextWatchDogTimer_bit0_AX_1" only has a constant 1'b1 value
50.                  : Assignment "arbiter. nextWatchDogTimer_bit0_AX_1" is defined on line 179 of "rr3.v"
51.
52.   Loss of valid data [LVD]   : 3 failed checks
53.   ---------------------------------------------
54.   FAILED    LVD : dataA[0]
55.                 : Loss of valid data has been detected on wire "dataA[0]"
56.                 : Wire "dataA[0]" is defined on line 40 of "rr3.v"
57.                 : VCD file is "roundRobin3/main/trace221 .vcd"
58.   FAILED    LVD :dataB[0]
59.                 : Loss of valid data has been detected on wire "dataB[0]"
60.                 : Wire "dataB[0]" is defined on line 41 of "rr3.v"
61.                 : VCD file is "roundRobin3/main/trace223.vcd"
62.   FAILED    LVD :dataC[0]
63.                 : Loss of valid data has been detected on wire "dataC[0]"
64.                 : Wire "dataC[0]" is defined on line 42 of "rr3.v"
65.                 : VCD file is "roundRobin3/main/trace225.vcd"
```

FIG. 13B

```
66.   ========================================
67.   Summary of bounded failed checks by type :
68.   ========================================
69.   [CA] =0  (out of 39)
70.   [BE] =0  (out of 40)
71.   [AX] =0  (out of 15)
72.   [CME]=0  (out of 10)
73.   [CV] =0  (out of 29)
74.   [AC] =0  (out of 7)
75.   [AAO] = 0  (out of 6)
76.   [AID]=0  (out of 9)
77.   [LVD] = 0  (out of 9)
78.
79.   ========================================
80.   Summary of inconclusive checks by type :
81.   ========================================
82.   [CA] =0  (out of 39)
83.   [BE] =0  (out of 40)
84.   [AX] =0  (out of 15)
85.   [CME]=0  (out of 10)
86.   [CV] =0  (out of 29)
87.   [AC] =0  (out of 7)
88.   [AAO]=0  (out of 6)
89.   [AID] = 0  (out of 9)
90.   [LVD] =3  (out of 9)
91.
92.   Loss of valid data [LVD] 3 inconclusive checks
93.   ----------------------------------------
94.   INCONCLUSIVE LVD   :busInterfaceA.dataOut[0]
95.                      : Valid data loss check on wire "busInterfaceA.dataOut[0]" did not complete
96.                      : Wire "busInterfaceA.dataOut[0]" is defined on line 80 of "rr3.v"
97.   INCONCLUSIVE LVD   :busInterfaceB.dataOut[0]
98.                      : Valid data loss check on wire "busInterfaceB.dataOut[0]" did not complete
99.                      : Wire "busInterfaceB.dataOut[0]" is defined on line 80 of "rr3.v"
100.  INCONCLUSIVE LVD   :busInterfaceC.dataOut[0]
101.                     : Valid data loss check on wire "busInterfaceC.dataOut[0]" did not complete
102.                     : Wire "busInterfaceC.dataOut[0]" is defined on line 80 of "rr3.v"
103.
104.  ========================================
105.  Summary of secondary failed checks by type :
106.  ========================================
107.  [CA] =0  (out of 39)
108.  [BE] =0  (out of 40)
109.  [AX] =0  (out of 15)
110.  [CME]=0  (out of 10)
111.  [CV] =0  (out of 29)
112.  [AC] =0  (out of 7)
113.  [AAO]=0  (out of 6)
114.  [AID]=0  (out of 9)
115.  [LVD]=0  (out of 9)
```

FIG. 13C

```
116.  ====================================
117.  Summary of interface checks by type :
118.  ====================================
119.  [CA] =0  (out of 39)
120.  [BE] =0  (out of 40)
121.  [AX] =0  (out of 15)
122.  [CME]=0  (out of 10)
123.  [CV] =0  (out of 29)
124.  [AC] =6  (out of 7)
125.  [AAO]=0  (out of 6)
126.  [AID] =3  (out of 9)
127.  [LVD]=3  (out of 9)
128.
129.  Assertion correctness [AC] : 6 interface checks
130.  ------------------------------------------------
131.  INTERFACE    AC      :busInterfaceA.envl
132.                       : Correctness check on assertion "busInterfaceA.envl" is at the interface
133.                       : Assertion "busInterfaceA.envl" is defined on line 121 of"rr3.v"
134.  INTERFACE    AC      :busInterfaceA.env2
135.                       : Correctness check on assertion "busInterfaceA.env2" is at the interface
136.                       : Assertion "busInterfaceA.env2" is defined on line 127 of "rr3.v"
137.                       : Check is used by conditional checks :
138.  AID                  : arbiter.dataA[0]
139.  INTERFACE    AC      :busInterfaceB.envl
140.                       : Correctness check on assertion "busInterfaceB.envl" is at the interface
141.                       : Assertion "busInterfaceB.envl" is defined on line 121 of "rr3.v"
142.  INTERFACE    AC      :busInterfaceB.env2
143.                       : Correctness check on assertion "busInterfaceB.env2" is at the interface
144.                       : Assertion "busInterfaceB.env2" is defined on line 127 of "rr3.v"
145.                       : Check is used by conditional checks :
146.  AID                  : arbiter.dataB[0]
147.  INTERFACE    AC      :busInterfaceC.envl
148.                       : Correctness check on assertion "busInterfaceC.envl" is at the interface
149.                       : Assertion "busInterfaceC.envl" is defined on line 121 of "rr3.v"
150.  INTERFACE    AC      :busInterfaceC.env2
151.                       : Correctness check on assertion "busInterfaceC.env2" is at the interface
152.                       : Assertion "busInterfaceC.env2" is defined on line 127 of "rr3.v"
153.                       : Check is used by conditional checks :
154.  AID                  : arbiter.dataC[0]
155.
156.  Access of invalid data [AID] : 3 interface checks
157.  ------------------------------------------------
158.  INTERFACE    AID     :dataA[0]
159.                       : Wire "dataA[0]" has accessed invalid data from the interface
160.                       : Wire "dataA[0]" is defined on line 40 of "rr3.v"
161.  INTERFACE    AID     :dataB[0]
162.                       : Wire "dataB[0]" has accessed invalid data from the interface
163.                       : Wire "dataB[0]" is defined on line 41 of "rr3.v"
164.  INTERFACE    AID     :dataC[0]
165.                       : Wire "dataC[0]" has accessed invalid data from the interface
166.                       : Wire "dataC[0]" is defined on line 42 of "rr3.v"
```

FIG. 13D

```
167.
168.  Loss of valid data [LVD] 3 interface checks
169.  ------------------------------------------
170.  INTERFACE   LVD        :arbiter.dataA[0]
171.                         : Wire "arbiter.dataA[0]" has loss of valid data at the interface
172.                         : Wire "arbiter.dataA[0]" is defined on line 141 of "rr3.v"
173.  INTERFACE   LVD        :arbiter.dataB[0]
174.                         : Wire "arbiter.dataB[0]" has loss of valid data at the interface
175.                         : Wire "arbiter.dataB[0]" is defined on line 142 of "rr3.v"
176.  INTERFACE   LVD        :arbiter.dataC[0]
177.                         : Wire "arbiter.dataC[0]" has loss of valid data at the interface
178.                         : Wire "arbiter.dataC[0]" is defined on line 143 of "rr3.v"
179.
180.  ========================================
181.  Summary of unprocessed checks by type :
182.  ========================================
183.  [CA] =0  (out of 39)
184.  [BE] =0  (out of 40)
185.  [AX] =0  (out of 15)
186.  [CME]=0  (out of 10)
187.  [CV] =0  (out of 29)
188.  [AC] =0  (out of 7)
189.  [AAO]=0  (out of 6)
190.  [AID]=0  (out of 9)
191.  [LVD]=0  (out of 9)
192.
193.  [...]
194.
195.  ==================================
196.  Summary of passed checks by type :
197.  ==================================
198.  [CA] =39  (out of 39)
199.  [BE] =39  (out of 40)
200.  [AX] = 14  (out of 15)
201.  [CME]= 10  (out of 10)
202.  [CV] =29  (out of 29)
203.  [AC] = 1  (out of 7)
204.  [AAO]=6  (out of 6)
205.  [AID] =6  (out of 9)
206.  [LVD] = 0  (out of 9)
207.
208.  [...]
209.
210.  ======================
211.  List of inferred flops
212.  ======================
213.  busInterfaceA.state[0]
214.  busInterfaceA.state[1]
215.  busInterfaceB.state[0]
216.  busInterfaceB.state[1]
217.  busInterfaceC.state[0]
218.  busInterfaceC.state[1]
219.  arbiter.state[0]
220.  arbiter.state[1]
221.  arbiter.state[2]
222.  arbiter.watchDogTimer[0]
```

FIG. 13E